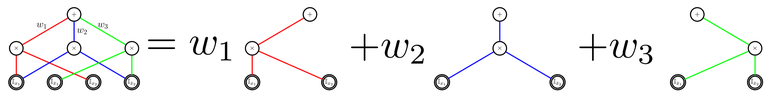
Linear Time Computation of Moments in Sum-Product Networks

Han Zhao and Geoff Gordon
Machine Learning Department, Carnegie Mellon University

Background

Sum-Product Networks (SPNs):

- Rooted directed acyclic graph of univariate distributions, sum nodes and product nodes.
- We focus on discrete SPNs, but the proposed algorithms work for continuous ones as well.



Recursive computation of the network:

$$V_k(\mathbf{x};\mathbf{w}) = \begin{cases} p(X_i = \mathbf{x}_i) & \text{if } k \text{ is a leaf node over } X_i \\ \prod_{j\in \mathrm{Ch}(k)} V_j(\mathbf{x};\mathbf{w}) & \text{if } k \text{ is a product node} \\ \sum_{j\in \mathrm{Ch}(k)} w_{k,j} V_j(\mathbf{x};\mathbf{w}) & \text{if } k \text{ is a sum node} \end{cases}$$

Scope: The set of variables that have univariate distributions among the node's descendants.
Complete: An SPN is *complete* iff each sum node has children with the same scope.
Decomposable: An SPN is *decomposable* iff for every product node v, $\mathrm{scope}(v_i) \cap \mathrm{scope}(v_j) = \varnothing$ where $v_i, v_j \in \mathrm{Ch}(v), i \neq j$.

Summary

- So far the best algorithm for moment computation scales quadratically in the size of Sum-product Networks (SPNs).
- We propose an optimal linear-time algorithm that works even when the SPN is a general directed acyclic graph (DAG).
  - A linear time reduction from the moment computation problem to a joint inference problem.
  - An efficient procedure for polynomial evaluation by differentiation without expanding the network.
  - A dynamic programming method to reduce the computation of the moments of all the edges from quadratic to linear.
- Applications:
  - Online moment matching
  - Assumed density filtering

Linear Time Exact Moment Computation

Exact Posterior Has Exponentially Many Modes:
Every complete and decomposable SPN $\mathcal{S}$ can be factorized into a sum of $\Omega(2^{H(\mathcal{S})})$ induced trees (sub-graphs), where each tree corresponds to a product of univariate distributions. $H(\mathcal{S})$ = height of $\mathcal{S}$:

$$p(\mathbf{w} \mid \mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \sum_{t=1}^{\tau} \prod_{k=1}^{m} \mathrm{Dir}(w_k; \alpha_k) \prod_{(k,j)\in\mathcal{T}_{tE}} w_{k,j} \prod_{i=1}^{n} p_t(x_i)$$

f-moment on each edge in the network:

$$\begin{aligned} M_p(f(w_{k,j})) &= \int_{\mathbf{w}} f(w_{k,j}) p(\mathbf{w} \mid \mathbf{x})\, d\mathbf{w} \\ &= \frac{1}{Z_{\mathbf{x}}} \sum_{t=1}^{\tau} c_t \int_{\mathbf{w}} p_0(\mathbf{w}) f(w_{k,j}) \prod_{(k',j')\in\mathcal{T}_{tE}} w_{k',j'}\, d\mathbf{w} \end{aligned}$$

- Naive computation: $\Omega(|\mathcal{S}| \cdot 2^{H(\mathcal{S})})$
- Recursive algorithm for trees: $O(|\mathcal{S}|)$, but $O(|\mathcal{S}|^2)$ for DAGs (Rashwan et al., 2016)

Linear Time Reduction to Joint Inference:
For each edge (k, j), partition all the trees into two sets:

- $\mathcal{T}_T = \{\mathcal{T}_t : t \in [\tau], (k,j) \in \mathcal{T}_t\}$, trees containing the edge.
- $\mathcal{T}_F = \{\mathcal{T}_t : t \in [\tau], (k,j) \notin \mathcal{T}_t\}$, trees not containing the edge.

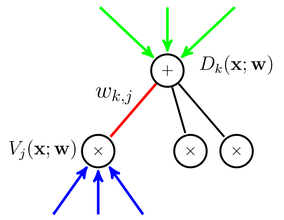
$$M_p(f(w_{k,j})) = \left(\frac{1}{Z_{\mathbf{x}}} \sum_{\mathcal{T}_t\in\mathcal{T}_F} c_t u_t\right) M_{p_{0,k}}(f(w_{k,j})) + \left(\frac{1}{Z_{\mathbf{x}}} \sum_{\mathcal{T}_t\in\mathcal{T}_T} c_t u_t\right) M_{p'_{0,k}}(f(w_{k,j}))$$

Efficient Polynomial Evaluation by Differentiation:
Key observation 1: $\sum_{t\in\mathcal{T}_T}^{\tau} c_t u_t + \sum_{t\in\mathcal{T}_F}^{\tau} c_t u_t$ can be computed in $O(|\mathcal{S}|)$ time and space in a bottom-up evaluation of $\mathcal{S}$, by re-defining the edge weights of the network.
Key observation 2: $\sum_{\mathcal{T}_t\in\mathcal{T}_T} c_t u_t = w_{k,j} (\partial \sum_{t=1}^{\tau} c_t u_t / \partial w_{k,j})$, and it can be computed in $O(|\mathcal{S}|)$ time and space in a top-down differentiation of $\mathcal{S}$, by the multilinear nature of the network polynomial.
Example:

$$\begin{aligned} & g(x_1, x_2, x_3) = 4x_1x_2 + 3x_2x_3 + 5x_1x_3 \\ \Rightarrow \quad & x_1 \frac{\partial}{\partial x_1} g(x_1, x_2, x_3) = 4x_1x_2 + 3x_2\cancel{x_3} + 5x_1x_3 \end{aligned}$$

Dynamic Programming

For each edge (k, j), moment can be computed in $O(|\mathcal{S}|)$ time and space. Naive computation leads to $O(|\mathcal{S}|^2)$ for all edges.



$$D_k(\mathbf{x};\mathbf{w}) = \frac{\partial V_{\mathrm{root}}(\mathbf{x};\mathbf{w})}{\partial V_k(\mathbf{x};\mathbf{w})}$$

Sufficient statistics for each edge contains three terms:

- Forward value V_j at the child.
- Backward value D_k at the parent.
- Edge weight $w_{k,j}$.

Quadratic to Linear: For all edges (k, j), moments can be computed in $O(|\mathcal{S}|)$ time, using only two more copies of the network.

Applications

Used as subroutines to develop linear time algorithms for Bayesian moment matching (BMM) and assumed density filters (ADF).
Runtime (log-seconds) over 20 real-world data sets.